**1. Top-Level Statements**

**Was sind Top-Level Statements (Anweisungen der obersten Ebene)?**

Top-Level Statements (**Anweisungen der obersten Ebene)** ermöglichen es, **Code direkt in der Datei ohne eine "Main-Methode"** zu schreiben.

Bis .NET 4 war eine sichtbare **Main-Methode** zwingend erforderlich:

class Program

{

static void Main(string[] args)

{

**Console.WriteLine("Hallo, Welt!");**

}

}

**Ab .NET 5** (Top-Level Statements) kann dies vereinfacht werden:

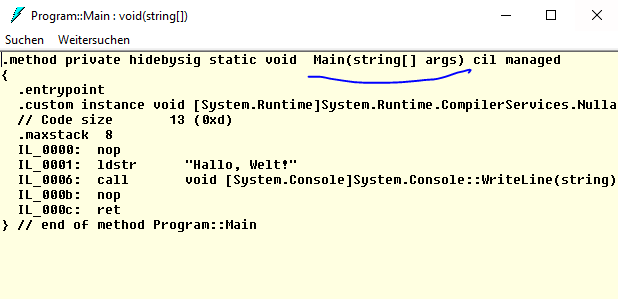
**Console.WriteLine("Hallo, Welt!");**

**Erklärung:**

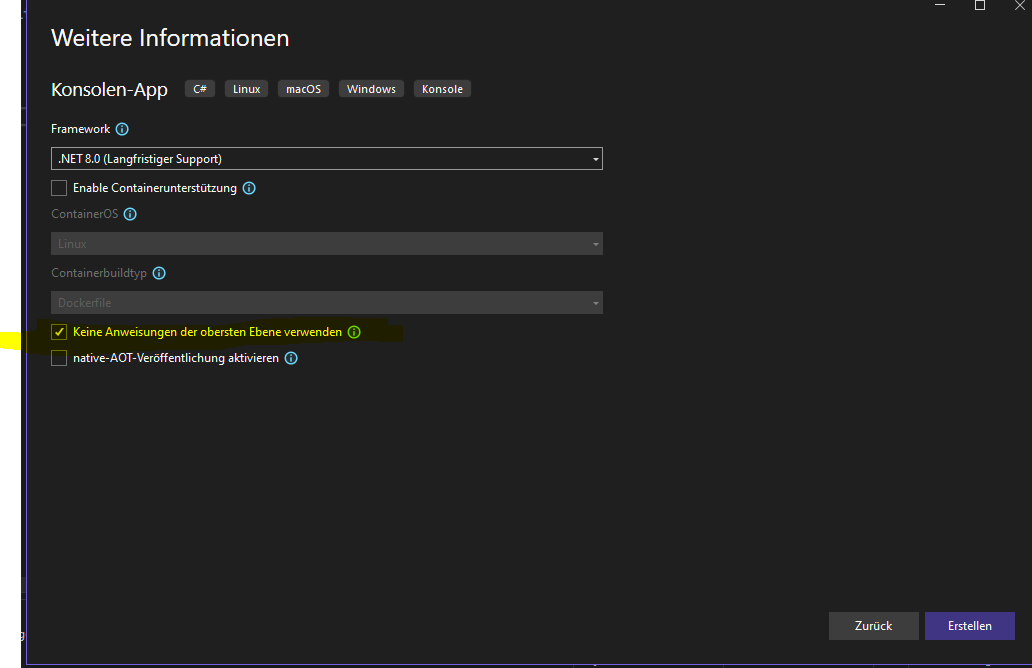
* C# erkennt automatisch den **Einstiegspunkt** des Programms. Der Compiler generiert eine **Program-Klasse** mit einer Einstiegspunktmethode und platziert alle Anweisungen der obersten Ebene in dieser Methode.
* Nützlich für kleine Skripte, Tutorials oder **minimalistische Konsolenanwendungen.**
* Diese „minimalistische Variante“ kann auch Namespaces (Namensräume) und Typdefinitionen (= Klassen) enthalten, diese müssen **aber nach den Anweisungen** definiert werden.
* Variable **args** wird automatisch erkannt
* **return-Anweisung** kann den Prozess unterbrechen

Im decompilierten MSIL-Code sieht man Main-Methode, die „automatisch“ erzeugt wurde.

C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.8 Tools\**ildasm.exe**



**Vorteile:**

* Weniger Segmente im Code.
* Einfacherer Einstieg für Anfänger.
* Übersichtlicher und klarer Code.
* Wie kann ich den **„alten“ Konsolen-Still verwenden**?

**2. Global Usings**

* **Global Usings** erlauben es, "using"-Anweisungen einmal für das gesamte Projekt zu definieren.
* Anstatt in jeder Datei **using System**; zu schreiben, wird es automatisch global eingebunden.
* De Begriff **implizite using-Anweisungen** bedeutet, dass der Compiler automatisch eine **Gruppe von using-Anweisungen** basierend auf dem Projekttyp hinzufügt. Für Konsolenanwendungen sind die folgenden Anweisungen **implizit** in der Anwendung enthalten:

using System;

using System.IO;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Threading;

using System.Threading.Tasks;

**Erklärung:**

* Aktivierung / Dekativierung in der .csproj-Datei: **<ImplicitUsings>enable</ImplicitUsings>**
* Implizite Usings kann man auch explizit ausschalten. Z.B.:   
  <ItemGroup>

<Using Remove="System.Net.Http" />

</ItemGroup>

**GlobalUsings.cs:**

**Zusätzliche** **Global Usings** kann man in einer **GlobalUsings.cs**-Datei definieren.

**Z.B.:**

**global** using System.Drawing;

**Vorteile:**

* Reduzierung der **Redundanz**: Keine wiederholten using-Anweisungen.
* **Übersichtlicherer Code**: Besonders bei großen Projekten mit vielen Klassen.
* **Einfachere Wartung**: Änderungen an using-Anweisungen müssen nur an einer Stelle vorgenommen werden.